Roll no- 391

Assignment No - 3

Name-Neha Dattatray Bhoite
Subject- SPOS

Program 1) Shortest Job First-

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.Scanner;
class Process {
   int id; // Process ID
   int arrivalTime; // Arrival time of the process
   int burstTime; // Burst time of the process
   int waitingTime; // Waiting time of the process
   int turnaroundTime; // Turnaround time of the process

   public Process(int id, int arrivalTime, int burstTime) {
      this.id = id;
      this.arrivalTime = arrivalTime;
      this.burstTime = burstTime;
   }
}
public class SJF {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter number of processes: ");
      int n = scanner.nextInt();

      Process[] processes = new Process[n];
      // Input process details
      for (int i = 0; i < n; i++) {
         System.out.print("Enter arrival time of process " + (i + 1) + ": ");
         int arrivalTime = scanner.nextInt();
         System.out.print("Enter burst time of process " + (i + 1) + ": ");
         int burstTime = scanner.nextInt();
         processes[i] = new Process(i + 1, arrivalTime, burstTime);
      }

      // Sort processes by arrival time
      Arrays.sort(processes, Comparator.comparingInt(p -> p.arrivalTime));
```

```
        // Calculate waiting time and turnaround time
        int currentTime = 0;
        int completed = 0;
        boolean[] isCompleted = new boolean[n];

        while (completed < n) {
            int idx = -1;
            int minBurstTime = Integer.MAX_VALUE;

            // Find the process with the smallest burst time that has arrived
            for (int i = 0; i < n; i++) {
                if (processes[i].arrivalTime <= currentTime && !isCompleted[i] &&
processes[i].burstTime < minBurstTime) {
                    minBurstTime = processes[i].burstTime;
                    idx = i;
                }
            }

            // If no process is available, increment current time
            if (idx == -1) {
                currentTime++;
            } else {
                // Update waiting and turnaround time for the selected process
                processes[idx].waitingTime = currentTime - processes[idx].arrivalTime;
                processes[idx].turnaroundTime = processes[idx].waitingTime +
processes[idx].burstTime;
                currentTime += processes[idx].burstTime;
                isCompleted[idx] = true;
                completed++;
            }
        }

        // Print results
        System.out.println("\nProcess ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround
Time");
        for (Process process : processes) {
            System.out.printf("%-12d\t%-12d\t%-12d\t%-12d\t%-12d\n",
                    process.id,
                    process.arrivalTime,
                    process.burstTime,
```

```
                process.waitingTime,
                process.turnaroundTime);
        }
            scanner.close();
    }
}
```

Output-

```
Enter number of processes: 4
Enter arrival time of process 1: 0
Enter burst time of process 1: 8
Enter arrival time of process 2: 1
Enter burst time of process 2: 4
Enter arrival time of process 3: 2
Enter burst time of process 3: 9
Enter arrival time of process 4: 5
Enter burst time of process 4: 3
```

Process ID	Arrival Time	Burst Time	Waiting Time	Turnaround Time
1	0	8	0	8
2	1	4	10	14
3	2	9	13	22
4	5	3	3	6

Program 2) First come First serve-

```
import java.util.Scanner;
class Process {
   int id; // Process ID
   int arrivalTime; // Arrival time of the process
   int burstTime; // Burst time of the process
   int waitingTime; // Waiting time of the process
   int turnaroundTime; // Turnaround time of the process

   public Process(int id, int arrivalTime, int burstTime) {
      this.id = id;
      this.arrivalTime = arrivalTime;
      this.burstTime = burstTime;
   }
}

public class FCFS {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter number of processes: ");
      int n = scanner.nextInt();

      Process[] processes = new Process[n];

      // Input process details
      for (int i = 0; i < n; i++) {
         System.out.print("Enter arrival time of process " + (i + 1) + ": ");
         int arrivalTime = scanner.nextInt();
         System.out.print("Enter burst time of process " + (i + 1) + ": ");
         int burstTime = scanner.nextInt();
         processes[i] = new Process(i + 1, arrivalTime, burstTime);
      }

      // Calculate waiting time and turnaround time
      int currentTime = 0;
      for (Process process : processes) {
         if (currentTime < process.arrivalTime) {
            currentTime = process.arrivalTime; // Wait for the process to arrive
```

```
            }
            process.waitingTime = currentTime - process.arrivalTime;
            currentTime += process.burstTime;
            process.turnaroundTime = process.waitingTime + process.burstTime;
        }

        // Print results
        System.out.println("\nProcess ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround
Time");
        for (Process process : processes) {
            System.out.printf("%-12d\t%-12d\t%-12d\t%-12d\t%-12d\n",
                    process.id,
                    process.arrivalTime,
                    process.burstTime,
                    process.waitingTime,
                    process.turnaroundTime);
        }
            scanner.close();
    }
}
```

Output-

```
Enter number of processes: 4
Enter arrival time of process 1: 0
Enter burst time of process 1: 5
Enter arrival time of process 2: 1
Enter burst time of process 2: 3
Enter arrival time of process 3: 2
Enter burst time of process 3: 8
Enter arrival time of process 4: 3
Enter burst time of process 4: 6

Process ID     Arrival Time    Burst Time      Waiting Time    Turnaround Time
1               0               5               0               5
2               1               3               4               7
3               2               8               6               14
4               3               6               13              19
```

Program 3) Round Robin Algorithm-

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

class Process {
   int id;
   int burstTime;
   int remainingTime;

   // Constructor
   Process(int id, int burstTime) {
      this.id = id;
      this.burstTime = burstTime;
      this.remainingTime = burstTime;
   }
}

public class RoundRobinScheduling {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the number of processes: ");
      int numberOfProcesses = scanner.nextInt();

      Process[] processes = new Process[numberOfProcesses];
      for (int i = 0; i < numberOfProcesses; i++) {
         System.out.print("Enter burst time for Process " + (i + 1) + ": ");
         int burstTime = scanner.nextInt();
         processes[i] = new Process(i + 1, burstTime);
      }

      System.out.print("Enter time quantum: ");
      int timeQuantum = scanner.nextInt();

      roundRobinScheduling(processes, timeQuantum);
   }

   public static void roundRobinScheduling(Process[] processes, int timeQuantum) {
```

```
        int n = processes.length;
        Queue<Process> queue = new LinkedList<>();

        int time = 0;
        int completedProcesses = 0;

        while (completedProcesses < n) {
            for (Process process : processes) {
                if (process.remainingTime > 0) {
                    queue.add(process);
                }
            }

            if (!queue.isEmpty()) {
                Process currentProcess = queue.poll();
                if (currentProcess.remainingTime > timeQuantum) {
                    time += timeQuantum;
                    currentProcess.remainingTime -= timeQuantum;
                    queue.add(currentProcess); // Re-add to the end of the queue
                } else {
                    time += currentProcess.remainingTime;
                    System.out.println("Process " + currentProcess.id + " completed at time " + time);
                    currentProcess.remainingTime = 0;
                    completedProcesses++;
                }
            }
        }
    }
}
```

Output-

```
PS C:\Users\DELL\Desktop\SPOS> javac RoundRobinScheduling.java
PS C:\Users\DELL\Desktop\SPOS> java RoundRobinScheduling
Enter the number of processes: 2
Enter burst time for Process 1: 3
Enter burst time for Process 2: 4
Enter time quantum: 3
Process 1 completed at time 3
Process 2 completed at time 7
```

Program 4) Priority Scheduling Algorithm-

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.Scanner;
class Process {
   int id;
   int burstTime;
   int priority;

   Process(int id, int burstTime, int priority) {
      this.id = id;
      this.burstTime = burstTime;
      this.priority = priority;
   }
}
public class PriorityScheduling {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the number of processes: ");
      int numberOfProcesses = scanner.nextInt();

      Process[] processes = new Process[numberOfProcesses];

      for (int i = 0; i < numberOfProcesses; i++) {
         System.out.print("Enter burst time for Process " + (i + 1) + ": ");
         int burstTime = scanner.nextInt();
         System.out.print("Enter priority for Process " + (i + 1) + ": ");
         int priority = scanner.nextInt();
         processes[i] = new Process(i + 1, burstTime, priority);
      }

      // Sort processes by priority (higher number means higher priority)
      Arrays.sort(processes, Comparator.comparingInt(p -> -p.priority));

      System.out.println("\nProcess Execution Order:");
      int time = 0;
```

```
        for (Process process : processes) {
            System.out.println("Process " + process.id + " (Burst Time: " + process.burstTime + ")
executed at time " + time);
            time += process.burstTime;
            System.out.println("Process " + process.id + " completed at time " + time);
        }
    }
}
```

Output-

```
Enter the number of processes: 1
Enter burst time for Process 1: 2
Enter priority for Process 1: 3

Process Execution Order:
Process 1 (Burst Time: 3) executed at time 0
Process 1 completed at time 3
```